ULTRIX

digital

Guide to X/Open curses Screen Handling

curses Screen Handling

Order Number: AA-LY27B-1E

ULTRIX

Guide to X/Open curses Screen Handling

Order Number: AA-LY27B-TE

June 1990

Product Version: ULTRIX Version 4.0 or higher

This manual describes the X/Open `curses` library routines. It describes the basic concepts of the library and shows how to write screen-management programs using the library routines.

digital equipment corporation
maynard, massachusetts

Contents

About This Manual

1 The X/Open Curses Library

2 Programming with the Curses Routines

Tables

About This Manual

This manual describes the X/Open `curses` library. It describes the basic concepts behind the library and explains how to write screen-management programs using the library routines.

The X/Open `curses` library routines coexist with the BSD 4.2 `curses` library routines. The X/Open routines are described in the `(3cur)` reference pages and are contained in the `libcursesX.a` library. The BSD 4.2 routines are described in the `(3x)` reference pages and are contained in the `libcurses.a` and `libtermlib.a` libraries. Both libraries deal only with character-cell displays such as character cell terminals or windows on a bit-map display that emulate character cell terminals.

Audience

This manual is intended for ULTRIX C programmers who want to find out about the X/Open curses library and how it can be used for writing screen-management programs for character cell displays. The documentation assumes that the reader understands the ULTRIX environment and knows how to program, compile, and link C code.

Organization

This manual contains two chapters, two appendixes, and an index. Read the chapters serially; refer to Appendix A and Appendix B as necessary.

Chapter 1	The X/Open curses Library Introduces the X/Open curses library, the basic concepts behind the library, and the individual library routines.
Chapter 2	Programming with the Curses Routines Describes the components needed to write a screen-management program and gives all the necessary background information.
Appendix A	Annotated Example Programs This appendix contains two screen-management programs written using curses routines. The programs are annotated to show how the curses routines function.
Appendix B	Comparison with BSD 4.2 Curses Routines This appendix outlines the differences between the two libraries of routines, and explains how to convert programs written using BSD 4.2 routines so they can use the X/Open routines.

Related Documents

Refer to the *ULTRIX Reference Pages, Section 3 (Library Routines)* for detailed descriptions of all the routines in the X/Open curses library. Start with the `intro`(3cur) reference page, which introduces the library and points you to other, related reference pages.

Conventions

The following text conventions are used in this document:

%	The default user prompt is your system name followed by a right angle bracket. In this manual, a percent sign (%) is used to represent this prompt.
`user input`	This bold typeface is used in interactive examples to indicate typed user input.
`system output`	This typeface is used in interactive examples to indicate system output and also in code examples and other screen displays. In text, this typeface is used to indicate the exact name of a command, option, partition, pathname, directory, or file.
UPPERCASE lowercase	The ULTRIX system differentiates between lowercase and uppercase characters. Literal strings that appear in text, examples, syntax descriptions, and function definitions must be typed exactly as shown.
macro	In text, bold type is used to introduce new terms.
filename	In examples, syntax descriptions, and function definitions, italics are used to indicate variable values; and in text, to give references to other documents.
. . .	A vertical ellipsis indicates that a portion of an example that would normally be present is not shown.
`cat`(1)	Cross-references to the *ULTRIX Reference Pages* include the appropriate section number in parentheses. For example, a reference to `cat`(1) indicates that you can find the material on the `cat` command in Section 1 of the reference pages.
RETURN	This symbol is used in examples to indicate that you must press the named key on the keyboard.
CTRL/*x*	This symbol is used in examples to indicate that you must hold down the CTRL key while pressing the key *x* that follows the slash. When you use this key combination, the system sometimes echoes the resulting character, using a circumflex (^) to represent the CTRL key (for example, ^C for CTRL/C). Sometimes the sequence is not echoed.

The X/Open Curses Library 1

This chapter contains the following topics:

- Definition of terms
- Introduction to the X/Open curses library
- Basic concepts
- Naming conventions
- Moving the cursor
- Environment variables
- Routines in the curses library

1.1 Definition of Terms

The following definitions are used in this manual:

Term	Definition
Window	This is an internal representation containing an image of what a section of the terminal screen may look like at some point in time. A window can be as big as the terminal screen or any smaller size, down to a single character.
Screen	This is a special case of window which represents the whole of the screen. The curses library provides a default screen, `stdscr`, to represent the screen.
Terminal	This is sometimes called the terminal screen and it is the physical screen. It is what the user actually sees on the screen at a particular time.
Subwindow	A subwindow is a new window created within an existing window. Any changes made to either the subwindow or to the subwindow area in the original window, are made in both windows.

1.2 Introduction to the Curses Library

The curses library is the X/Open set of library routines used for writing screen-management programs. The X/Open set of library routines provides additional

features to the BSD 4.2 set of curses routines. See Appendix B for a comparison of the two sets of routines.

The curses library enables C programmers to do the common types of terminal-dependent functions without worrying about the detailed description of the current terminal. The routines also save programming time by making it easy to describe how a program should update screens.

Screen-management programs are concerned both with updating screens and moving the cursor in an efficient way.

The name `curses` is derived from the term **cursor optimization.** Optimizing cursor motion means minimizing the amount the cursor has to move to update the screen.

All the curses routines are located in the curses library, `/usr/lib/libcursesX.a`. You do not need to use any other routines for writing screen-management programs.

You can use the curses library to write interactive screen-management programs. Some of the tasks performed by screen-management programs are:

- Writing output to, and reading input from, a terminal screen
- Dividing a terminal screen into windows
- Sending output to, and accepting input from, more than one terminal
- Moving the cursor in the most efficient way
- Drawing a display on the screen for data entry and retrieval
- Displaying modified screen layouts

The curses library is split into two parts:

- Screen updating, both output and with user input
- Cursor-motion optimization

The screen-updating functions are used when parts of a screen need to be changed but the overall image remains the same. For example, consider a screen that shows data input fields. When data is input to a particular field and the RETURN key is pressed, the screen-management program updates only that particular field. The rest of the screen layout is not updated. This process is also known as **output optimization,** since the output to the screen is optimized.

The cursor motion part of the library can be used separately from the screen-updating routines. Cursor-motion optimization is used on its own for tasks such as defining how the cursor moves in response to tabs and newline characters.

1.3 Basic Concepts

To update a screen efficiently, the curses program must have information on what the current display on the screen looks like and how the programmer wants that display to appear next.

The `<cursesX.h>` header file defines a data type (structure) called `WINDOW`, which is used by the curses routines as a representation of the screen. The definition includes the starting position of the window on the screen and the window size. The curses routines write to this internal representation of the terminal screen instead of writing directly on to the physical screen.

A window can be thought of as a two-dimensional array of characters on which to build and store a potential image of the terminal screen. The window can represent all or part of a terminal screen. You can create smaller subwindows in existing windows called **pads,** which are bigger than the actual terminal screen. Pads are used for applications which do not need to show all of the window on the screen at any one time, for example, an application using a spreadsheet.

The curses library maintains a record of all the characters on the screen at all times. Two default windows for this purpose are:

- `stdscr` – A representation of the terminal screen on which to make changes.
- `curscr` – The current image on the terminal screen. Not usually accessed directly by the application.

Both of these default windows represent the whole of the user's terminal screen.

The application makes changes to the default window `stdscr` or to a named window. The changes made to the window are not transferred to the physical screen until the application calls the `refresh`(3cur) routine.

The screen-management program keeps track of what is on `stdscr` and what is on the screen. When it gets a call to `refresh`(3cur), it compares these two images, and sends a stream of characters to the terminal to make the physical screen look like `stdscr`. To find the optimal way of doing this, the screen-management program takes into account the following factors:

- The capabilities of the terminal
- The similarities between what is on the screen and what is on the window

At the most basic level, `stdscr` is manipulated by the routines `move`(3cur), which moves the cursor around the screen, and `addch`(3cur), which adds characters to the screen.

An application may use these routines to add data to the window in any convenient order. Higher level routines combine the actions of `move`(3cur) and `addch`(3cur). There are also routines to add strings and to convert formatting information in the same way as `printf`(3s).

Multiple new windows can be created using `newwin`(3cur), which allows an application to build several images of the screen and to quickly display the appropriate one. For example, one window can control input/output and another can display error messages.

There are also routines to do the following:

- Erase the entire window
- Specify the video attributes of individual characters in the window
- Open additional terminals by large applications that need to manipulate several terminals at once
- Allow input character manipulation
- Disable and enable many input attributes

There is more detailed information about these routines in Section 1.7 of this chapter.

1.4 Naming Conventions

Many of the curses routines have two or more forms depending on whether they apply to the default window `stdscr`, to a specific named window, or if cursor movement is involved.

The routines without a prefix to their names normally manipulate `stdscr`. For many of these routines there is a corresponding routine name prefixed with a `w` to manipulate the contents of a specified window; for example, `move`(3cur) and `wmove`(3cur), which are functionally equivalent. This naming convention is similar to the `stdio`(3s) interface offered by `printf` and `fprintf`.

Routines prefixed with `p` require a `pad` argument.

Routines prefixed with `w` require a `window` or `pad` argument, except for `wnoutrefresh`(3cur) and `wrefresh`(3cur), which only accept a `window` argument. In these two cases, `prefresh`(3cur) or `pnoutrefresh`(3cur) must be used if a pad is to be manipulated.

Routines prefixed with `mv` require `y` and `x` coordinates to move to, before performing the appropriate action. The `mv` routines call `move`(3cur) before the call to the other routine.

The routines prefixed with `mvw` require a `window` or `pad` argument and `y` and `x` coordinates. The `window` argument (the pointer to the window) is always specified before the coordinates.

1.5 Moving the Cursor

All the routines that move the cursor move it from the home position in the upper left corner of the screen. The *LINES, COLS* coordinates at this point are (1,1). Note that the vertical coordinate `y` is given first and the horizontal coordinate `x` is given second. The upper left corner of the window is always (0,0), not (1,1). Consequently the command `move(y,x)`, with `y` = 1 and `x` = 0, will move the window cursor to the second line, first column of the screen. Note that specified coordinates are always relative to the home position (1,1), which is the first position on the screen that can be written to.

1.6 Environment Variables

You can employ the environment variables to define or modify definitions for display-related values. See `environ`(7) for more information. The routines in the library recognize the following environment variables:

- *TERM* sets the terminal type.
- *TERMINFO* overrides the default `terminfo`(5) database pathname, `/usr/lib/terminfo`.
- *LINES* overrides the default set for the number of lines for the display by `stty` or by the TERM environment variable.
- *COLUMNS* overrides the default set for the number of columns for the display by `stty` or by the *TERM* envrionment variable.

You can override the environment variables supplied by `environ`(7) for *LINES* and *COLUMNS* specifying the LINES and COLS general constants in your application. See Section 2.1.2. for more information about the general constants.

Two ULTRIX facilities exist to help prepare and maintain the X/Open curses terminals database. They are the `terminfo`(5) terminal capability database and the `tic`(1) terminal database compiler.

1.7 Routines in the Curses Library

This section lists most of the curses routines organized according to function. The routines prefixed with `mv` and `mvw` are not included in this list as they perform the same functions as equivalent routines which are listed; the only difference is that they require x and y coordinates to move to before performing the appropriate action. The naming conventions for the routines are described in Section 1.4.

The curses routines are divided into the following functional groups:

- Setting screen characteristics
- Window manipulation
- Adding characters to windows (output to windows)
- Clearing windows and deleting characters
- Refreshing the terminal screen
- Input to windows
- Input options
- Output options
- Environment queries and terminal characteristics
- Miscellaneous routines
- Starting and ending curses programs

An alphabetical list of all the curses routines, with their required arguments, appears in Appendix B.

The following sections describe the routines which manipulate `strscr`. Where there is an equivalent routine which manipulates a named window, this routine name is shown in parentheses.

1.7.1 Setting Screen Characteristics

These routines control the attributes of characters displayed on the screen. For more information on attributes, see Section 2.1.3. Routines prefixed with a `w` for example, `wattroff`, are used to manipulate the contents of a particular window.

The following routines are used to set screen attributes such as highlighting:

attroff (wattroff)	Turns off the named attributes, attrs
attron (wattron)	Turns on the named attributes, attrs
attrset (wattrset)	Sets the current attributes to the named attributes

setscrreg (wsetscrreg)	Sets the scrolling region for stdscr
standend (wstandend)	Switches off the highlighting mode available on the terminal
standout (wstandout)	Switches on the best highlighting mode available on the terminal
vidattr	Outputs a string that sets the video attributes for the terminal

1.7.2 Window Manipulation

The following routines are used to create, delete, and move windows:

delwin	Deletes a named window.
movewin	Moves a window, with upper left corner at given coordinates.
newpad	Creates a new pad.
newwin	Creates a new window.
overlay	Copies text from one window to another. Blanks are not copied so this does not destroy all the contents of the original window.
overwrite	Copies all of one window on top of another window. Blanks are copied as well so this destroys all the contents of the original window.
subwin	Creates a subwindow.

1.7.3 Adding Characters to Windows (Output to Windows)

The following routines either move the cursor to a specified position and add characters to a window, or they add characters at the current cursor position:

addch (waddch)	Inserts a character into stdscr at current cursor position.
mvaddch (mvwaddch)	Moves the cursor to a specified position and inserts a character.
addstr (waddstr)	Writes the characters of a null-terminated character string on to stdscr at the current cursor position.

mvaddstr (wmvaddstr)	Writes the characters of a string to a specified position on the screen.
printw (wprintw)	Adds a string to stdscr at the current cursor position. The printw routines are analogous to prinft(3s).
mvprintw (mvwprintw)	Adds a string to stdscr starting at the specified cursor position.
insch (winsch)	Inserts a character at the current cursor position in stdscr.
insertln (winsertln)	Inserts a blank line above the current line on stdscr.
mvinsch (wmvinsch)	Moves the cursor to specified position in stdscr and inserts a character.
move (wmove)	Moves the cursor associated with stdscr to a specified position.

1.7.4 Clearing Windows and Deleting Characters

The following routines are used to clear windows or parts of windows, and to delete individual characters and lines:

clear (wclear)	Resets the whole of stdscr to blanks, and sets the current (y,x) coordinates to (0,0). On the next call to refresh(3cur), the terminal screen is cleared.
clrtobot (wclrtobot)	Begins at the current cursor position and clears the rest of the screen to blanks.
clrtoeol (wclrtoeol)	Erases the current line from the cursor onwards.
clearok	If true is specified, screen clearing is enabled. If a window is specified, the next call to `wrefresh` completely clears that window.
delch (wdelch)	Deletes the character under the cursor at the current cursor position.
mvdelch (wmvdelch)	Moves the cursor to a specified position and deletes the character there.
deleteln (wdeleteln)	Deletes the whole of the line that the cursor is on.
erase (werease)	Copies blanks to every position on stdscr.

1.7.5 Refreshing the Terminal Screen

The characters on a window are transferred to the screen when one of the following routines is called to refresh it:

prefresh	Copies a named pad to the terminal screen
pnoutrefresh	Copies a named pad to stdscr, compares it with the terminal screen, and then optimally updates the terminal screen
refresh	Copies stdscr to the terminal screen
wrefresh	Calls wnoutrefresh
wnoutrefresh	Copies a named window to stdscr, compares it with the terminal screen, then calls doupdate to update the terminal screen in an efficient way
leaveok	Allows the cursor to be left wherever the update happens to leave it

Note that there is no definition for the routine `doupdate`, as it is a routine which is called by other routines, and is not called directly by the programmer.

1.7.6 Input to Windows

The following routines are used to read characters from the current terminal as input for `stdscr` or a named window:

getch (wgetch)	Reads a character from the terminal associated with stdscr
mvgetch (mvwgetch)	Reads a character from the specified position on stdscr
getstr (wgetstr)	Reads characters from the terminal associated with stdscr, until a newline or carriage return is received
mvgetstr (mvwgetstr)	As getstr, but moves the cursor to a specified position in stdscr to read a string

1.7.7 Input Options

The interpretation of characters typed by the user can be controlled by the following `curses` routines:

cbreak	The erase/kill characters typed by the user are not interpreted
nocbreak	Disable cbreak mode
echo	Enables echoing of characters typed by the user
noecho	Disables echoing
flushinp	Discards any typeahead that has not been read by the program
nl	Enables the newline character to be translated to a newline on input
nodelay	Enables getch to be a non-blocking call (it does not return an error if there are no characters waiting)
nonl	Disables nl on input
raw	In raw mode, the characters are passed to the program as they are typed and there is no interpretation of control characters
noraw	Disables raw mode
typeahead	If enabled, the program accepts typeahead input while updating the screen

1.7.8 Output Options

The output to the screen can be controlled by the following routines:

beep	Sounds the audible alarm on the terminal, otherwise flashes the screen
flash	Flashes the screen if possible, otherwise it sounds the audible alarm
delay_output	Causes a short delay in output which can be specified (in milliseconds)
draino	Waits until there is a specified (milliseconds) amount of output left in the output queue
intrflush	If enabled, flushes all output in the tty driver queue when an interrupt key is pressed
napms	Causes a program to sleep for a specified number of milliseconds
nl	When enabled, causes a newline to be translated into a carriage return and a line-feed on output

nonl	Disables nl
unctrl	Expands a character into a character string which is a printable representation of the character, for example, control characters are expressed as ^X

1.7.9 Environment Queries and Terminal Characteristics

The following routines are used for getting information about the current terminal characteristics:

baudrate	Returns the terminal baudrate
erasechar	Returns the user's current erase character
getyx	Gets the cursor position
has_ic	Determines if the terminal has insert/delete line capabilities
has_il	Returns the value TRUE if the terminal has insert/delete line capabilities
idlok	Enables the use of insert/delete line if the terminal has these capabilities
inch (winch)	Returns a character at the current cursor position in stdscr
mvinch (mvwinch)	Returns a character at a specified position in stdscr
keypad	Enables the terminal keypad so that the function keys can return a single value to the curses program
longname	Returns the full name of the terminal type
meta	On input, forces the user's terminal to return 7 significant bits if FALSE or 8 significant bits if TRUE

1.7.10 Miscellaneous Routines

The following routines offer additional functions for curses programs:

def_prog_mode	Saves current terminal modes as the *program* if the terminal is running under curses. Used by the `reset_prog_mode` routine when the user makes a temporary exit from the curses program.
def_shell_mode	Saves current terminal modes as the *shell* if the terminal is not running under curses. Used by the `reset_shell_mode` routine.
mvcur	Performs low-level cursor movement.
newterm	Opens a new terminal.
setupterm	Performs low-level terminal set up.

1.7.11 Starting and Ending Curses Programs

The routine `initscr` must be called at the beginning of all curses programs to initialize the terminal. The routine `endwin` must be called at the end of every curses program to restore the terminal to the state it was in before running the curses program.

initscr	Initializes all terminal data structures
endwin	Restores initial terminal environment after exiting from the curses program

Programming with the Curses Routines 2

This chapter covers the following topics:

- The <cursesX.h> header file
- Return values
- Terminfo and the curses package
- Basic program elements
- Controlling input
- Compiling programs
- Restrictions on terminals

2.1 The <cursesX.h> Header File

The curses library package supports a procedural interface to all of the defined data types, so the actual structures of the data types are not described. All the curses data is manipulated using the routines provided by the curses library package.

The data types are defined in the `<cursesX.h>` header file which must always be included whenever X/Open curses routines are used in a program. The `<cursesX.h>` header file also includes the header file `<stdio.h>` which uses the standard Input/Output library.

The `<cursesX.h>` header file also defines various global constants and the combinations of routines which make up the curses macros. General constants, video attribute constants, and input constants are described later in this chapter.

2.1.1 Data Types

The following data types are declared:

Data Type	Description
WINDOW*	Pointer to the screen representation, stdscr
SCREEN*	Pointer to terminal descriptor, curscr
bool	Boolean data type
chtype	Representation of a character in a window

The actual `WINDOW` and `SCREEN` objects used to store information are created by the corresponding routines and a pointer to them is provided. All manipulation is through that pointer. The data type `chtype` contains both data and attributes for an individual character.

2.1.2 General Constants

The following general constants are defined:

Constant	Description
COLS	Number of columns on terminal screen
ERR	Value returned on error condition
FALSE	Boolean *false* value
LINES	Number of lines on terminal screen
OK	Value returned on successful completion
NULL	Zero pointer value
TRUE	Boolean *true* value

The integer variables `LINES` and `COLS` are set up so that when a curses program is run on a particular terminal, these variables are assigned the vertical and horizontal dimensions of the current terminal screen. The routine `initscr(3cur)` is used for assigning these dimensions.

2.1.3 Video Attribute Constants

The window `stdscr` has a set of current attributes which it associates with each character as it is written. The current attributes can be changed by using `attrsett`(3cur) and related routines such as `attron`(3cur) and `attroff`(3cur). The attributes can also be `OR`ed with the bitwise `OR` (|) to `addch(3cur)`. The following constants (and the attributes they define) can be passed to the appropriate curses routines:

Constant	Description
A_BLINK	Blinking
A_BOLD	Extra bright or bold
A_DIM	Half bright
A_REVERSE	Reverse video
A_STANDOUT	Terminal's best highlighting mode
A_UNDERLINE	Underlining
A_ATTRIBUTES	Bit-mask to extract attributes
A_CHARTEXT	Bit-mask to extract a character

Not all terminals are capable of displaying all attributes. If a terminal can not display a requested attribute, a curses program attempts to find a substitute. If no substitute is available, then the attribute is ignored. An attribute can be used on its own or in combination with other attributes.

The characters passed to some of the curses routines are of the type `chtype`, as defined in the `<cursesX.h>` header file. This data type contains both data and attributes for an individual character.

2.1.4 Input Constants

When keypad(3cur) is enabled, and you press a function key such as the left arrow key, the routine getch(3cur) returns a single value, representing the function key, to the program. For example, when you press the left arrow key, the keypad(3cur) routine returns a value of KEY_LEFT to the program. The <cursesX.h> header file contains the definitions of possible function keys. All of the definitions begin with KEY_.

If the curses program receives a character that could be the beginning of the sequence for a function key, it sets a timer. If it does not receive the rest of the sequence for the function key within the designated time, the character is passed to the program as a single character. If the rest of the sequence does arrive, the value for the function key is returned. This explains why on many terminals there is a delay in returning escape to a program after the escape key has been pressed.

If keypad(3cur) is disabled, the curses program does not treat function keys as special keys.

Table 2-1 shows the constants that can be returned by getch(3cur) if keypad(3cur) is enabled. Note that some of the function keys in Table 2-1 are not supported on a particular terminal if:

- The terminal does not transmit a unique code when the key is pressed
- The definition for the key is not present in the underlying table of terminal capabilities

2.1.5 The Virtual Keypad

The virtual keypad is arranged in the following way:

A1	up	A3
left	B2	right
C1	down	C3

The layout of a keypad is terminal dependent, especially the part associated with the cursor movement keys (the arrow keys). The curses package provides a set of generic keys, for example KEY_A1, which is defined in this example as the upper left key of the virtual keypad.

The code sequence transmitted when a particular function key is pressed, depends on how the terminal has been set up. A key in a particular virtual position can transmit different code sequences depending on which terminal is being used. The curses program finds out about the function keys by using information provided by the terminfo database and routines (see Section 2.3 for more information).

2.2 Return Values

Unless there is a note to the contrary in the reference page descriptions, the following return values apply:

- All routines return the value `OK` upon successful completion
- All routines return the value `ERR` on failure
- Routines that return pointers always return the NULL pointer on error

The keypad return constants are shown in Table 2-1. All these constants are defined in the `<cursesX.h>` header file (see Section 2.1).

Table 2-1: Keypad Return Constants

Constant	Description
KEY_BREAK	Break key
KEY_DOWN	Down arrow key
KEY_UP	Up arrow key
KEY_LEFT	Left arrow key
KEY_RIGHT	Right arrow key
KEY_HOME	Home key (upward+left arrow)
KEY_BACKSPACE	Backspace
KEY_F0	Function keys; space is reserved for 64 keys
KEY_F(n)	(KEY_F0+(n))
KEY_DL	Delete line
KEY_IL	Insert line
KEY_DC	Delete character
KEY_IC	Insert character or enter insert mode
KEY_EIC	Exit insert character mode
KEY_CLEAR	Clear screen
KEY_EOS	Clear to end of screen
KEY_EOL	Clear to end of line
KEY_SF	Scroll 1 line forward
KEY_SR	Scroll 1 line backwards (reverse)
KEY_NPAGE	Next page
KEY_LPAGE	Previous page
KEY_STAB	Set tab
KEY_CTAB	Clear tab
KEY_CATAB	Clear all tabs
KEY_ENTER	Enter or send
KEY_SRESET	Soft (partial) reset
KEY_RESET	Reset or hard reset
KEY_PRINT	Print or copy
KEY_LL	Home down or bottom (lower left)
KEY_A1	Upper left of virtual keypad
KEY_A3	Upper right of virtual keypad
KEY_B2	Center of virtual keypad
KEY_C1	Lower left of virtual keypad
KEY_C3	Lower right of virtual keypad

2.3 Terminfo and the Curses Package

This section provides the following:

- What information is needed about the current terminal
- Where that information can be found
- How the curses routines use that information

The curses routines update the screen in a way appropriate for the terminal on which the program is running. The programmer does not need to know the detailed terminal characteristics of the different terminal types that can be updated.

The curses program searches the `terminfo` database to find the correct description for a terminal.

2.3.1 What Is Terminfo?

The term **terminfo** applies to two things:

- A group of routines within the curses library that handle certain terminal capabilities
- A database containing descriptions of many terminals that can be used with curses programs

The terminfo routines can be used to program function keys, if the terminal has programmable keys.

The terminfo database contains descriptions of many terminals that can be used with curses programs. The database contains information such as the number of lines and columns on a terminal screen and how control characters are interpreted. Each terminal description in the database is a separate compiled file.

2.3.2 How Curses and Terminfo Work Together

A screen-management program written using curses routines needs certain information about the terminal on which it is currently running. The program gets the terminal type from the environment variable `TERM`. You can supply `TERM` when you log in, or you can set it up and export it in your `.profile` file.

When the program finds the value of `TERM` it searches the terminfo database to find the correct terminal description. For example, if the standard library for the database is `/usr/lib/terminfo` and `TERM` is set to `vt100`, then the compiled file will be normally be found in `/usr/lib/terminfo/v/vt100`. The directory name `v` is copied from the first letter of vt100 to avoid creation of huge directories.

If the environment variable `TERMINFO` is defined, any program using curses routines will check for a local terminal definition before checking in the standard libraries.

For example, if `TERMINFO` is set to `/usr/mark/myterms`, then the curses program will first check `/usr/mark/myterms/v/vt100`, and if that fails will check `/usr/lib/terminfo/v/vt100`. This facility is useful for developing experimental definitions or when write permission is not available in `/usr/lib/terminfo`.

The terminfo routines are very low level and their use in programs is not encouraged. There is more information in the `intro`(3cur) reference page if it is required.

2.4 Basic Program Elements

A screen-management program needs to include the following elements:

- The <cursesX.h> header file
- The routine initscr(3cur)
- The routine refresh(3cur) or other related routines
- The routine endwin(3cur)

These basic elements respectively do the following tasks:

- Start the screen-handling process
- Update the contents of the screen in the most efficient way
- Exit from the screen-handling routines

The program uses the environmental variable TERM to determine the type of terminal being used.

2.4.1 The <cursesX.h> Header File

The <cursesX.h> header file defines various global constants and declares the data types which are available to an application. The default window, stdscr, which is the same size as the current terminal screen, is also provided by the header file.

Note that you can override the variables set up in the header file by defining in your program your own environment variables such as LINES and COLS. This is useful if you want to change the official size of a terminal screen.

Note

You can set default sizes for windows (the lines and columns) by setting the runtime environment variables *LINES* and *COLUMNS*. See Section 1.6 for more information about these environ(7) environment variables.

2.4.2 The Routine initscr(3cur)

The curses program calls the routine initscr(3cur) to allocate memory space for the windows. However, it should call this routine only once as it can overflow available memory if it is called repeatedly. The routine returns ERR if this happens.

Once the routine has allocated memory space, it initializes all the declared data structures and other variables from the <cursesX.h> header file, writes any error messages to stderr and exits if errors occur.

You should always call the routine to initialize the terminal before calling any of the routines that operate on windows. If possible, you should call the routine after checking for start-up errors. However, be sure to call any routines that change the status of the terminal after you call initscr(3cur).

Note

The first call to `initscr`(3cur) after changing the official size of the screen deletes the default windows `stdscr` and `curscr` before creating new ones. As a consequence, if these present default windows are important, you should change the size of the screen by adjusting the variables `LINES` and `COLS` before this first call to `initscr`(3cur).

2.4.3 Routines Providing Output for Writing to stdscr

The following routines add data to windows. The curses program does not transfer any of the data onto the terminal screen until the routine `refresh`(3cur) is called.

addch(3cur)	Writes one character to stdscr
addstr(3cur)	Adds a character string to stdscr
move(3cur)	Moves the cursor and prints characters
printw(3cur)	Formats a string

Note that for each routine that acts on `stdscr` there is a related routine in the library for writing to a named window or pad.

2.4.4 Routines for Clearing Windows

The following routines clear all or part of a window. The terminal screen is not affected until the program calls `refresh`(3cur).

clear(3cur)	Clears the default window to blanks and sets the current (y,x) coordinates to (0,0)
erase(3cur)	Copies blanks to every position in the default window
cltoeol(3cur)	Clears to the end of the cursor line
clrtobot(3cur)	Clears to end of screen

Note that for each routine that acts on `stdscr` there is a related routine in the library for clearing a named window.

2.4.5 Routines for Reading from the Current Terminal

The following routines provide input to a window from the current terminal:

getch(3cur)	Reads one character at a time from the terminal associated with the default window stdscr
getstr(3cur)	Reads a string terminated by a carriage return from the terminal associated with stdscr

scanw(3cur)	Parses input, converting and assigning selected data to an argument list

Note that for each routine which reads from the terminal associated with `stdscr` there is a related routine which reads characters from the terminal associated with a specified window.

2.4.6 Updating the Physical Terminal Screen

The routine `refresh`(3cur) is called to update the physical terminal screen. This routine takes into account what is already on the terminal screen in order to optimize cursor movement to update the screen.

The routine moves the cursor back to the window's current (y,x) co-ordinates after it has updated the window. Use `leaveok`(3cur) if you want to leave the cursor in its last position when you update the window.

The routine uses the contents of `stdscr` to update the terminal screen. There are other routines for using the contents of specific windows and pads to update the terminal screen (see Section 1.7.5).

2.4.7 Ending a Curses Program

The routine `endwin`(3cur) restores all terminal settings to what they were prior to running the curses program and positions the cursor at the lower left corner of the screen.

Note

You should not use Input/Output routines or system calls from other libraries in a curses program. The curses library provides its own set of input and output functions which support procedural interfaces to all the data types defined in the `<cursesX.h>` header file. If you try to use other Input/Output routines or system calls such as `read`(2) and `write`(2) in a curses program, they may cause undesirable results when you run the program.

2.5 Controlling Input

While a curses program is running, it takes over the standard ULTRIX character mode processing in order to maintain total control over the screen.

Normal character mode processing of a character occurs before the character is passed to an application. The processing includes the following features:

- Echoing characters to the terminal as they are typed
- Interpreting the erase line and line kill characters
- Interpreting a CTRL/D as the end of file
- Interpreting the interrupt and quit characters
- Stripping a character's parity bit
- Translating a carriage return to a newline

The curses program turns echoing off and takes over all the echoing itself, in order to maintain total control over the screen. The routines noecho(3cur) and cbreak(3cur) change the standard character processing. They are used to control how input is interpreted. The routine noecho(3cur) turns off echoing at the current cursor position and echoes characters at the bottom of the screen. The routine cbreak(3cur) turns off the interpretation of erase and kill characters.

A curses program always starts up in echo mode, but the other modes must be specifically set up if they are required.

Most interactive screen programs need character-at-a-time input without echoing. To achieve this, the following routines should be called at the beginning of a curses program:

```
nonl();
cbreak();
noecho();
```

The routine nonl(3cur) disables the newline control translations so that a carriage return is not translated into a newline on input.

2.6 Compiling Programs

You can compile a curses program using the cc(1) command. Use the -l option to direct the link editor to the curses library.

The syntax for compiling programs written using curses routines is:

```
cc [options] files -lcursesX [libraries]
```

A specific example of compiling the file, curses_prog.c using the cursesX library would be:

```
cc curses_prog.c -lcursesX
```

2.7 Restrictions on Terminals

Some restrictions may apply when writing applications used for driving synchronous, networked asynchronous or non-standard directly connected asynchronous terminals.

These terminals often communicate with the host in block-mode, which means that characters are not transmitted to the host one at a time as they are typed. In block mode, the user types characters at the terminal then presses a special key to initiate transmission of all the characters to the host.

However, it may not be possible or desirable to cause a character to be transmitted with only a single keystroke. Single keystroke character transmission can cause severe problems with an application using single character input (see Section 2.7.2).

2.7.1 Output

The curses package can be used in the normal way for all output operations to the terminal, with the possible exception that on some terminals the refresh(3cur) routine may have to redraw the entire screen contents in order to perform any update.

2.7.2 Input

Because of the nature of operation of synchronous (block-mode) and networked asynchronous terminals, it may not be possible to support all or any of the curses input functions. In particular, the following points should be noted:

- Single character input may not be possible. It may be necessary to press a special key to cause all characters typed at the terminal to be transmitted to the host.
- It may not be possible to disable echo. Character echo may be performed directly by the terminal.

On terminals that perform character echo, programmers writing curses applications which get input from the terminal should be aware that any characters typed will appear on the screen at the physical cursor position. This may not necessarily correspond to the position of the cursor in the window.

Annotated Example Programs A

This appendix contains two short example programs. The programs are annotated to show the functions of the different routines as they are called.

A.1 Example Program 1

This short program displays the word MIDSCREEN in the center of the terminal screen. To illustrate the various steps, the word is added in two stages.

```
/* Include the header file */
#include <cursesX.h>
main ()
{
/* Initialize terminal settings, data
 **structures and variables */
initscr();

/* Move the cursor to given coordinates on
 **stdscr */
move (LINES/2 -1, COLS/2 -4);

/* Add the string "MID" to stdscr */
addstr("MID")

/* Send output from stdscr to update
 **terminal screen */
refresh();

/* Add the string "SCREEN" to stdscr. stdscr now
 ** contains the whole string "MIDSCREEN",
 **but the terminal screen only
 **shows "MID" */
addstr("SCREEN")

/* Send more output to the terminal screen
 **from stdscr */
refresh();

/* Restore all terminal settings to what they
were before the curses program ran */
endwin();
}
```

A.2 Example Program 2

This program displays an asterisk at a random point on the screen, waits for a space to be typed, and loops. Input is read one character at a time, with echo turned off.

```
/*
** stars.c
**   curses demonstration program
*/

/* Include the header file */
#include <cursesX.h>

#include <signal.h>

extern void srand();
extern void exit();
/*
** trap()
** invoked on receipt of interrupt signal
** reset terminal modes and exit
*/
trap(sig)
int sig;
{
        (void) endwin();
        exit(sig);

}

main()
{
int x;
int y;

/* Trap interrupts */
(void) signal(SIGINT, trap);

/* Initialize terminal */
(void) initscr();

/* Set terminal for character at a time
 ** input without echoing */
(void) noecho();
(void) cbreak();
(void) clear();

/* Seed the random number generator */
srand((unsigned) getpid());

     /* Loop */
     for (;;) {

                /* Generate random
                 **coordinates */
                y = rand() % LINES;
            x = rand() % COLS;

                /* Move cursor to given
                 **coordinates */
                void)move(y, x);

                /* Add '*' to stdscr */
                (void) addch('*');
```

```
            /* Update terminal screen */
            (void) refresh();

            /* Wait for space to be typed */
            while (getch() != ' ');

        }

         /* not reached */
}
```

Comparison with BSD 4.2 Curses Routines B

This appendix contains the following:

- A list of the major differences between the X/Open and the BSD 4.2 routines
- Information on how to convert programs using BSD curses routines so that they can use X/Open curses routines
- An alphabetical list of all the X/Open routines side by side with the BSD routines

B.1 Differences Between the X/Open Routines and the BSD 4.2 Routines

The X/Open routines coexist with the BSD 4.2 routines which were written before the requirement that routines conform to X/Open standards. The X/Open routines retain all the functionality of the BSD routines but with the following additional features:

- The addition of the `mv` functions such as `mvaddch`(3cur)
- Eight-bit data transparency for internationalization
- Compatibility with AT&T System V, Release 2

B.2 Converting BSD 4.2 Programs to Use X/Open Routines

It is possible to convert programs which use BSD 4.2 routines so that they can use the X/Open routines. Programs written using X/Open routines cannot use the BSD routines.

Programs written using the BSD 4.2 routines use the `/usr/lib/libtermcap.a` library. Some routines are included in the X/Open library as a conversion aid for programs that have used the BSD 4.2 termcap library. These routines have the same parameters as those in the termcap library, and are emulated using the `terminfo` database. The routines provided for conversion purposes are:

Routine	Function
`tgetent(bp,name)`	Look up termcap entry for name
`tgetflag(id)`	Get boolean entry for id
`tgetnum(id)`	Get numeric entry for id
`tgetstr(id,area)`	Get string entry for id
`tgoto(cap,col,row)`	Apply parms to given cap
`tputs(cap,affcnt,fn)`	Apply padding to cap calling fn as putchar

Note

These routines are not defined in X/Open and should not be used in new programs using the X/Open curses routines.

To convert a BSD 4.2 curses program, compile it using the `<cursesX.h>` header file.

To aid compatibility between the two sets of curses routines, the object module `termcap.o` has been provided in `/usr/lib/termcap.o`. This module should be linked into an application before resolving against the X/Open curses library. If your application references variables, such as `UP`, then you should also recompile using:

```
cc [flags] files /usr/lib/termcap.o -lcursesX [libs]
```

B.3 List of BSD 4.2 Routines and X/Open Routines

Table B-1 lists the BSD 4.2 routines alpabetically with the corresponding X/Open routines. For more information on the naming conventions used for the curses routines, see Section 1.4.

To save repetition in the table, the functions prefixed with `mv` or `mvw` are not defined. These routines move the cursor and then perform the same functions as the routines which are described for `stdscr` or a named window. For example, the routine `mvaddch` moves the cursor and adds a character to `stdscr` in the same way as `addch`. The routine `mvwaddch`, moves the cursor to the specified coordinates in a named window and adds a character in the same way as `addch`.

Table B-1: BSD 4.2 Routines and X/Open Routines

BSD Curses	X/Open Curses	Function
addch(ch)	addch(ch)	Adds a character to stdscr
addstr(str)	addstr(str)	Adds a string to stdscr
	attroff(attrs)	Turns off named attributes
	attron(attrs)	Turns on named attributes
	attrset(attrs)	Sets current attributes to attrs
	baudrate()	Displays current terminal speed
	beep()	Sounds beep on terminal
box (win,vert,hor)	box (win,vert,hor)	Draws a box around a window
	cbreak()	Sets cbreak mode
clear()	clear()	Clears stdscr

Table B-1: (continued)

BSD Curses	X/Open Curses	Function
clearok (scr,boolf)	clearok (win,bf)	Sets clear flag for stdscr
clrtobot()	clrtobot()	Clears to bottom of stdscr
clrtoeol()	clrtoeol()	Clears to end of line on stdscr
crmode()	see cbreak	Sets cbreak mode
	delay_output(ms)	Inserts millisecond pause (ms) in output
delch()	delch()	Deletes a character
deleteln()	deleteln()	Deletes a line
delwin(win)	delwin(win)	Deletes *win*
echo()	echo()	Sets echo mode
endwin()	endwin	Ends window modes
erase()	erase	Erases stdscr
	erasechar	Returns user's erase character
	fixterm()	Restores terminal to its ''in curses'' state
	flash()	Flashes screen or beep
	flushinp	Throws away any typeahead
getch()	getch()	Gets a char through tty
getcap(name)		Gets the terminal capability
getstr(str)	getstr()	Gets a string from stdscr
gettmode()	gettmode	Establishes current tty modes
getyx (win,y,x)	getyx (win,y,x)	Gets (y,x) co-ordinates
	has_ic()	True if terminal can do insert character
	has_il()	True if terminal can insert line

Table B-1: (continued)

BSD Curses	X/Open Curses	Function
	idlok(win,bf)	Uses terminal's insert/delete line if bf !=0
inch()	inch()	Gets char at current (y,x) co-ordinates
initscr()	initscr()	Initializes screens
insch(c)	insch(c)	Inserts a character
insertln()	insertln()	Inserts a line
	intrflush (win,bf)	Interrupt flushes output if bf is TRUE
	keypad(win,bf)	Enables keypad input
	killchar()	Returns user's current kill character
leaveok(win,boolf)	leaveok(win,flag)	Leaves cursor anywhere after refresh, if flag !=0 for *win*. Otherwise, cursor must be left at current cursor position
longname (termbuf,name)	longname()	Returns verbose name of terminal
	meta(win,flag)	Allows meta characters on input if flag !=0
move(y,x)	move(y,x)	Moves the cursor to (y,x) on stdscr
	mvaddch (y,x,ch)	
	mvaddstr (y,x,str)	
	mvcur(oldrow, oldcol,newrow, newcol) newx)	Low level cursor motion
	mvdelch(y,x)	
	mvgetch(y,x)	
	mvgetstr(y,x)	

Table B-1: (continued)

BSD Curses	X/Open Curses	Function
	mvinch(y,x)	
	mvinsch(y,x,ch)	
	mvprintw (y,x,fmt,args)	
	mvscanw (y,x,fmt,args)	
	mvwaddch (win,y,x,ch)	
	mvaddstr (win,y,x,str)	
	mvwdelch (win,y,x)	
	mvwgetch (win,y,x)	
	mvgetstr (win,y,x)	
	mvwin (win,by,box)	
	mvwinsch (win,y,x,ch)	
	mvwprint (win,y,x,fmt,args)	
	mvwscanw (win,y,x,fmt,args)	
	newpad (nlines,ncols)	Creates a new pad with given dimensions
	newterm (type,fd)	Sets up a new terminal of given type to output on fd
newwin	newwin (lines,cols, begin_y,begin_x)	Creates a new window (lines,cols begin_y,begin_x)
nl()	nl()	Sets newline mapping
	nocbreak()	Unsets cbreak mode

Table B-1: (continued)

BSD Curses	X/Open Curses	Function
nocrmode()	see nocbreak	Unsets cbreak mode
noecho()	noecho()	Unsets echo mode
nonl()	nonl()	Unsets newline mapping
noraw()	noraw()	Unsets raw mode
overlay	overlay (win1, win2)	Overlays window1 on window2 (win1,win2)
overwrite (win1,win2)	overwrite (win1,win2)	Overwrites window1 on top of window2
	pnoutefresh (pad,pminrow, pmincol,sminrow, smincol,smaxrow, smaxcol)	Like prefresh but with no output until doupdate called
	prefresh (pad,pminrow, pmincol,sminrow, smincol,smaxrow, smaxcol)	Refreshes from pad, starting with given upper left corner of pad with output to given portion of screen
printw (fmt,arg1...)	printw (fmt,arg1...)	The same as printf(3s) on stdscr
raw()	raw()	Sets raw mode
refresh()	refresh()	Makes current screen look like stdscr
	resetterm()	Sets tty modes to its out of curses state
resetty()	resetty()	Resets tty flags to stored value
	saveterm()	Saves current modes as the in curses state
savetty()	savetty()	Stores current tty flags
scanw (fmt,arg1...)	scanw (fmt(arg1...)	The same as scanf(3s) through stdscr
scroll(win)	scroll(win)	Scrolls *win* one line
scrollok	scrollok	Allows terminal to scroll if flag !=0

Table B-1: (continued)

BSD Curses	X/Open Curses	Function
(win,boolf)	(win,flag)	
setterm(name)	see set_term	Sets term variables for name
	set_term(new)	Switches between different variables
	setsccreg(t,b)	Sets user scrolling regions to lines t through b
	setupterm (term,filenum, erret)	Low level terminal setup
standend()	standend()	Ends standout mode
standout()	standout()	Starts standout mode
subwin(win, lines,cols, begin_y,begin_x)	subwin(win, lines,cols, begin_y,begin_x)	Creates a subwindow
touchwin(win)	touchwin(win)	Changes all of the specified window
	traceoff()	Turns off debugging trace output
	traceon()	Turns on debugging trace output
	typeahead(fd)	Uses file descriptor fd to check typeahead
unctrl(ch)	unctrl(ch)	Produces printable version of *ch*
waddch(win,ch)	waddch(win,ch)	Adds a character to *win*
waddstr(win,str)	waddstr(win,str)	Adds a string to *win*
	wattroff(win,attrs)	Turns off attributes in *win*
	wattron(win,attrs)	Turns on attributes in *win*
	wattrset(win,attrs)	Sets attributes in *win*
wclear(win)	wclear(win)	Clears *win*
wclrtobot(win)	wclrtobot(win)	Clears to bottom of *win*

Table B-1: (continued)

BSD Curses	X/Open Curses	Function
wclrtoeol(win)	wclrtoeol(win)	Clears to end of line on *win*
wdelch(win,c)	wdelch(win)	Deletes a character from *win*
wdeleteln(win)	wdeleteln(win)	Deletes a line from *win*
werase(win)	werase(win)	Erases *win*
wgetch(win)	wgetch(win)	Gets a char from *win*
wgetstr (win,str)	wgetstr (win,str)	Gets a string from *win*
winch(win)	winch(win)	Gets a character at current (y,x) in named window
winsch (win,c)	winsch (win,ch)	Inserts a character into window
winsertln (win)	winsertln (win)	Inserts line into window
wmove (win,y,x)	wmove (win,y,x)	Gets current (y,x) co-ordinates on window
	wnoutrefresh (win)	Refreshes but no screen output
wprintw (win,fmt, arg1,arg2..)	wprintw (win,fmt, arg1,arg2...)	The same as printf(3s) on *win*
wrefresh(win)	wrefresh(win)	Makes screen look like stdscr
wscanw (win,fmt, arg1,arg2,...)	wscanw (win,fmt, arg1,arg2...)	The same as scanf(3s) through *win*
	wsetscrreg (win,t,b)	Sets scrolling region of *win*
wstandend (win)	wstandend (win)	Ends standout mode on *win*
wstandout (win)	wstandout (win)	Starts standout mode on *win*

Index

How to Order Additional Documentation

Technical Support

If you need help deciding which documentation best meets your needs, call 800-343-4040 before placing your electronic, telephone, or direct mail order.

Electronic Orders

To place an order at the Electronic Store, dial 800-234-1998 using a 1200- or 2400-baud modem from anywhere in the USA, Canada, or Puerto Rico. If you need assistance using the Electronic Store, call 800-DIGITAL (800-344-4825).

Telephone and Direct Mail Orders

Your Location	Call	Contact
Continental USA, Alaska, or Hawaii	800-DIGITAL	Digital Equipment Corporation P.O. Box CS2008 Nashua, New Hampshire 03061
Puerto Rico	809-754-7575	Local Digital Subsidiary
Canada	800-267-6215	Digital Equipment of Canada Attn: DECdirect Operations KAO2/2 P.O. Box 13000 100 Herzberg Road Kanata, Ontario, Canada K2K 2A6
International	———	Local Digital subsidiary or approved distributor
Internal*	———	SSB Order Processing - WMO/E15 *or* Software Supply Business Digital Equipment Corporation Westminster, Massachusetts 01473

* For internal orders, you must submit an Internal Software Order Form (EN-01740-07).

Reader's Comments

ULTRIX
Guide to X/Open curses
Screen Handling
AA-LY27B-TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

Please rate this manual:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

What would you like to see more/less of? _____

_____

_____

What do you like best about this manual? _____

_____

_____

What do you like least about this manual? _____

_____

_____

Please list errors you have found in this manual:

Page	Description
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____

Additional comments or suggestions to improve this manual:

_____

_____

_____

_____

What version of the software described by this manual are you using? _____

Name/Title _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Email _____ Phone _____

Do Not Tear – Fold Here and Tape

digital™

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST–CLASS MAIL PERMIT NO. 33 MAYNARD MA

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
OPEN SOFTWARE PUBLICATIONS MANAGER
ZKO3–2/Z04
110 SPIT BROOK ROAD
NASHUA NH 03062–9987

Do Not Tear – Fold Here

Cut
Along
Dotted
Line

Reader's Comments

ULTRIX
Guide to X/Open curses
Screen Handling
AA-LY27B-TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

Please rate this manual:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

What would you like to see more/less of? _____

_____

_____

What do you like best about this manual? _____

_____

_____

What do you like least about this manual? _____

_____

_____

Please list errors you have found in this manual:

Page	Description
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____

Additional comments or suggestions to improve this manual:

_____

_____

_____

_____

What version of the software described by this manual are you using? _____

Name/Title _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Email _____ Phone _____

Do Not Tear – Fold Here and Tape

digital™

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST–CLASS MAIL PERMIT NO. 33 MAYNARD MA

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
OPEN SOFTWARE PUBLICATIONS MANAGER
ZKO3–2/Z04
110 SPIT BROOK ROAD
NASHUA NH 03062–9987

Do Not Tear – Fold Here

Cut
Along
Dotted
Line

digital